RESUME SCREENING USING VARIOUS MODELS

Linqing Zhu, Hyeran Park

Introduction

Resume screening is the process of reviewing a resume to determine if the candidate is qualified for the position. With thousands of resumes, selecting a proper candidate is time-consuming and it can be a challenging task for companies. This screening process is finding similar patterns matching based on past records or classifying (categorizing) resumes with similar traits. Our approach to the problem of resume screening is to create a resume screening system to classify resumes. We expect to accept resume pdf files as input and use Optical Character Recognition (OCR) model to extract text from the resume pdf and get the keywords in the resume by using natural language processing. Then categorize and match them with job descriptions or roles ideally looking for. This solution will be beneficial for companies who want to look for a good candidate from thousands of resumes online and categorize resumes to promote the job.

Part I. Resume Dataset 1

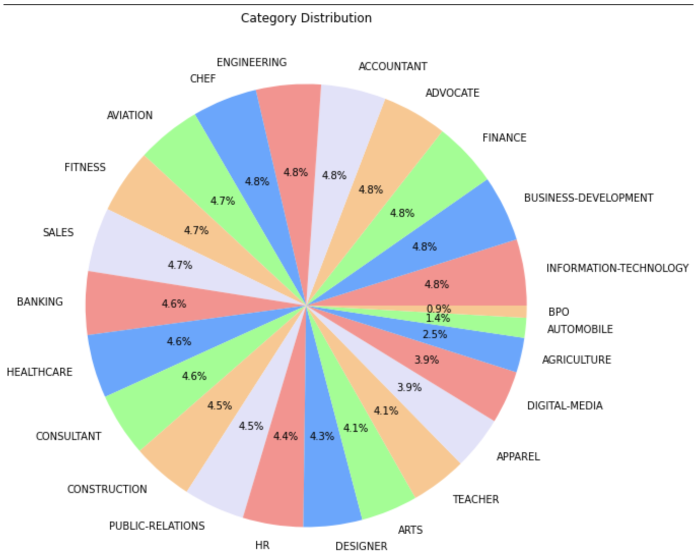
I-1. Method

(a) Dataset

This project utilized *Resume Dataset* [1]from Kaggle, which consists of 962 resumes with information in two columns: "Category" and "Resume". The "Category" column has 25 unique values, and the distribution of categories is relatively even. The categories include: 'Advocate', 'Arts', 'Automation Testing', 'Blockchain', 'Business Analyst', 'Civil Engineer', 'Data Science', 'Database', 'DevOps Engineer', 'DotNet Developer', 'ETL Developer', 'Electrical Engineering', 'HR', 'Hadoop', 'Health and fitness', 'Java Developer', 'Mechanical Engineer', 'Network Security Engineer', 'Operations Manager', 'PMO', 'Python Developer', 'SAP Developer', 'Sales', 'Testing', and 'Web Designing'.

Additionally, we used a resume in pdf format to experiment with image processing techniques. The resume has the current job position, skill sets, and name of a person who works in the technology field.

[1] https://www.kaggle.com/datasets/gauravduttakiit/resume-dataset



(b) Data Cleaning and Preprocessing Steps

Resumes are often in PDF format, which makes it easier to use an optical character recognition (OCR) model to extract the text. We considered three OCR options: Tesseract, EasyOCR, and Keras-OCR. Tesseract performs well with high-resolution images and can be enhanced through the use of morphological operations such as dilation, erosion, and OTSU binarization. EasyOCR is a lightweight model that performs well with structured texts like PDFs, receipts, and bills. Keras-OCR is an image-specific OCR tool that is effective for extracting text from unstructured images with irregular fonts and colors. We ultimately chose to use Tesseract because it is better suited for reading clear images with alphabetic characters.

The "Resume Dataset" contained unnecessary characters such as '*, (,), â¢' and others. To prepare the dataset for use with OCR models, we needed to remove these characters and only retain alphabetic characters or words. We used the Python re (Regular Expression) library to clean the resume data with a custom function. This function removed URLs, RT, cc, hashtags, mentions, punctuations, whitespace, and new lines.

After cleaning the "Resume Dataset", we transformed the "Category" values into numerical numbers using a label encoder, as the original values were nominal data. We vectorized the strings using TfidfVectorizer with the "cleaned_resume" column as X and "Category" as Y. We chose to use the Tfidf model because it not only counts the occurrences of words in a document, like the Bag of Words model, but it also takes into account the importance of each word. Finally, we split the dataset into the train, validate, and test sets with a ratio of 80:10:10 and used the shuffle function for the RNN model.

(c) Machine Learning Methods

In a study by Divya Mule, Samiksha Doke, Shakshi Navale, and Prof S.K.Said (2022), the authors proposed using LSTM, KNN, and Support Vector Machine methods for resume screening. We decided to use the Bidirectional LSTM and KNN methods because they are suitable for multiclass classification. Bidirectional LSTM (BiLSTM) is a type of recurrent neural network that is commonly used in natural language processing. It differs from standard LSTM in that it processes input in both directions, allowing it to utilize information from both sides of the sequence. BiLSTM is effective at modeling the sequential dependencies between words and phrases in both directions of the sequence.

I-2. Experiments

(a) KNN

To train the KNN model, we used both datasets with the default parameter k of 5 and a range of 1 to 32. We utilized cross-validation with 5 folds to find the best parameter k and assess the accuracy and mean squared error. We then summarized the evaluation results in terms of accuracy and f1 score using the model summary and visualized the results to compare and evaluate the accuracy.

(b) RNN (Bidirectional LSTM)

After preprocessing the data, we built the model with an embedding dimension of 64 and experimented with different activation functions, including ReLu and Tanh. Since this was a multiclass classification task, we set the activation function for the output layer to softmax.

We also attempted to combine the categories into three broader categories: 'Tech', 'Non_Tech', and 'Engineering'. We applied the same models to this modified dataset and compared the results.

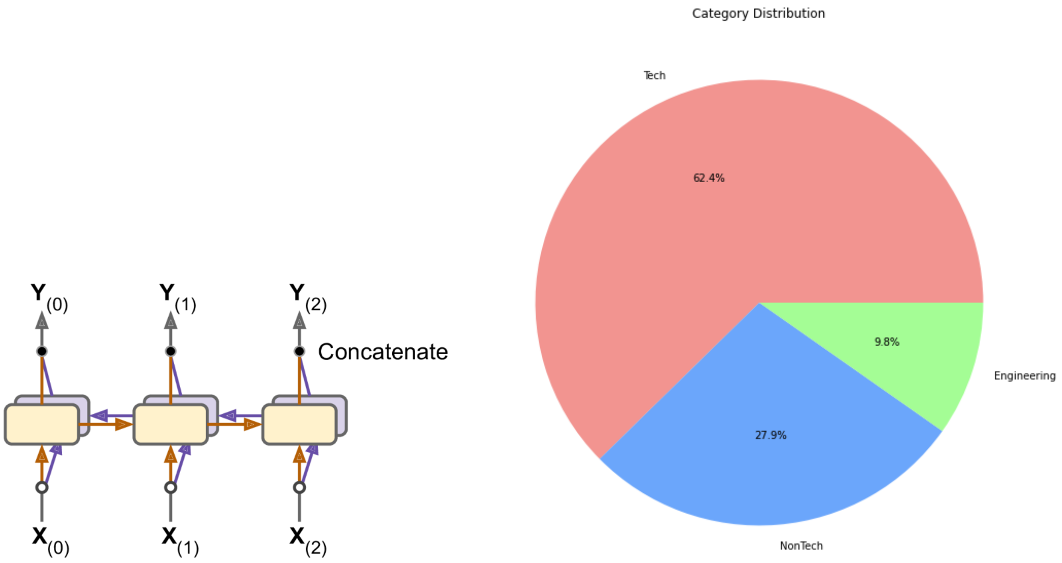


Fig I-2.(b) Bidirectional RNN concept figure(Left) Combined Categories figure(right)

I-3. Results and discussion

3.1 KNN

From the first dataset KNN approach, we found that the ‘ResumeDataset.csv’ file (the number of 962 records) has really high accuracy when the hyperparameter k is small. Especially when the k value is 1 or 2, we observed the accuracy is almost 99%.

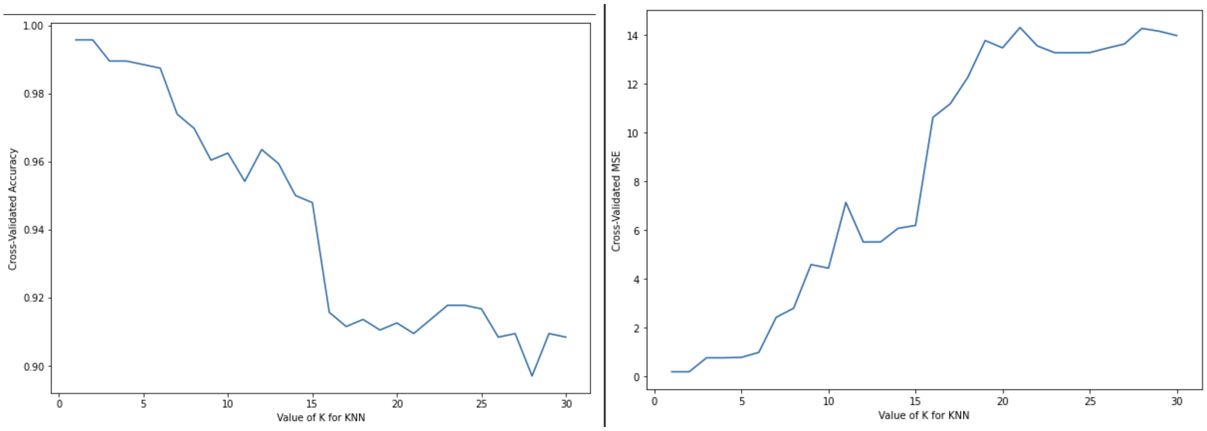


Fig I-3.1-1 Cross-validation accuracy and MSE of original data fig

From the cross-validation in accuracy and the mean squared error results(fig I-3.1-1). It seems like the accuracy is unreasonably high in the low k_neighbors. There could be data leakage or the model is classifying tech vs non-tech resumes.

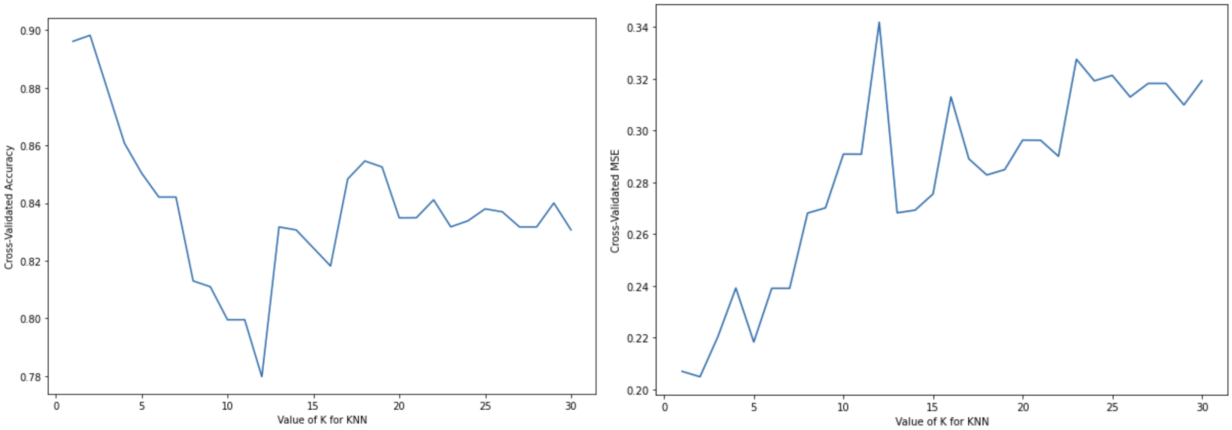


Fig I-3.1-2 Cross-validation accuracy and MSE of combined categories

The result based on the combined categories has a more sharp velocity. However, the result shows that setting the parameter k as 1 or 2 will generate high accuracy and a low MSE number. Since approximately 60% of the data is combined in the ‘Tech’ category, and the remaining 40% consists of only 2 classes, it is suspected that after 2 or 3 of clustering, accuracy goes down rapidly and the error rate increases with a steep slope.

3.2 RNN

For the bidirectional RNN model with the original dataset, we achieved a high accuracy of around 91-95% (Table I-3.2-2). Between the Tanh and ReLU activation functions, Tanh performed slightly better than ReLU. After 10 epochs, the accuracy was almost 100%, so we decided to stop training at that point. The text normalization process involved removing unnecessary punctuation and performing tokenization. We also found that the embedding dimension had a significant impact on the accuracy of the test data. When we decreased it to approximately half (64 to 30), the accuracy decreased to 0.7113401889801025.

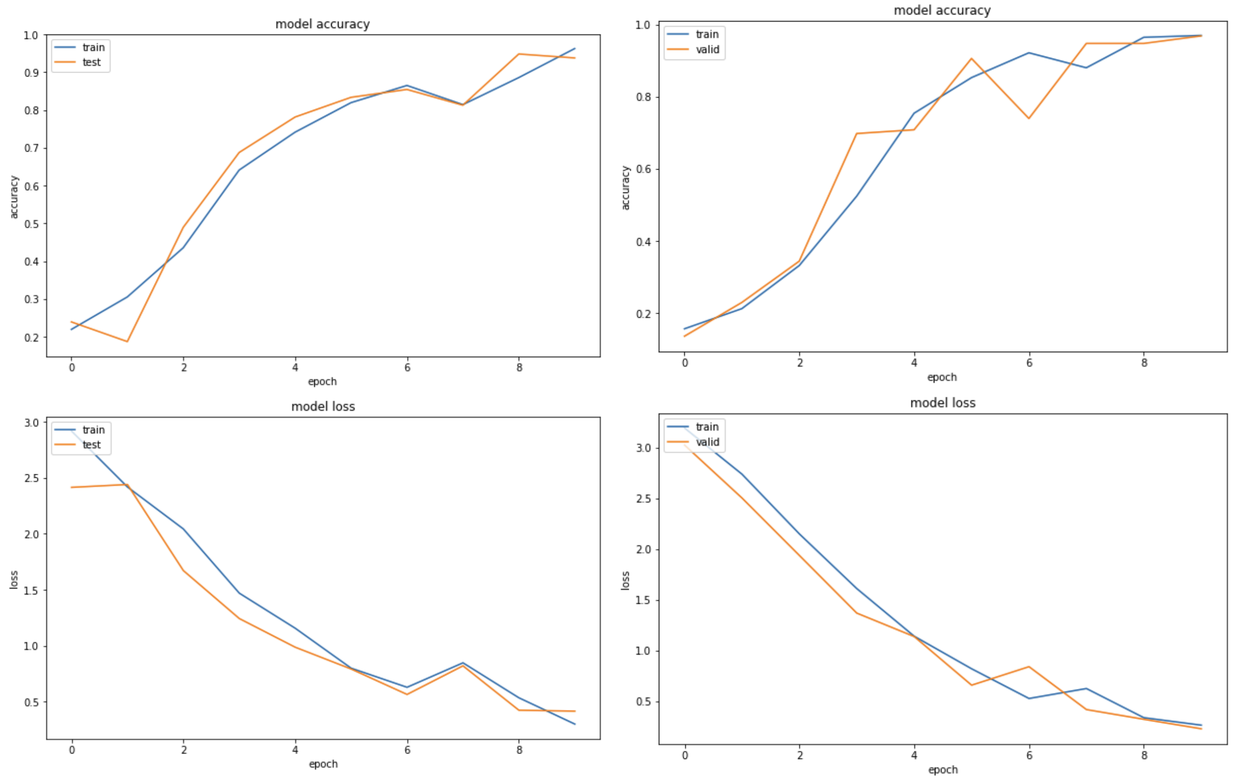


Fig I-3.2-1 Accuracy and loss when using ReLU(left) and Tanh(right)

Original Data	ReLU	Tanh
Test Score	0.4663737118244171	0.2864454984664917
Test Accuracy	0.9175257682800293	0.9587628841400146

Table I-3.2-2 Test score and accuracy between ReLU and Tanh

After combining categories into 3 broad categories, the performance of the RNN model was not as promising. We observed that the training accuracy was already reaching over 90% and the loss was very small in the early epochs, indicating a potential for overfitting. The testing results showed very low accuracy (Table I-3.2-4). We experimented with different embedding dimensions but saw only a slight improvement in the test value (0.1% to 0.15%). The unbalanced distribution of the split dataset was a suspected reason for the poor performance, but we found that the distribution of the three categories in the test, train, and validation sets was similar.

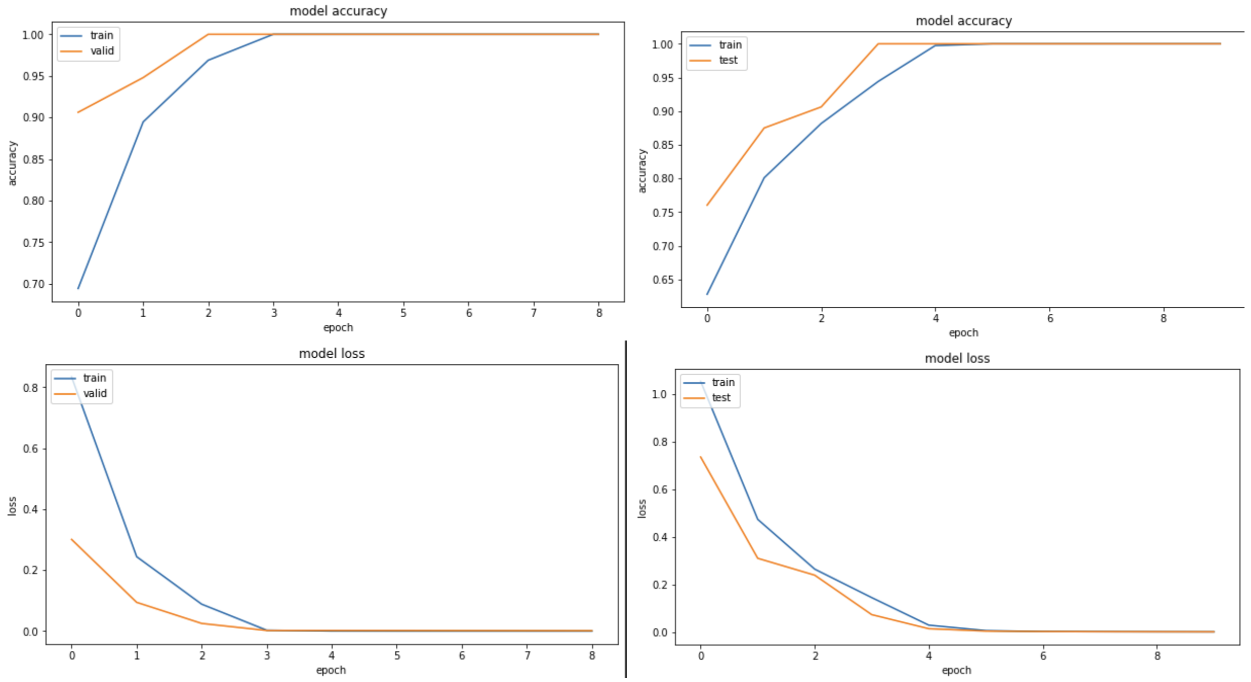


Fig 3.2-3 Train and Test accuracy and loss ReLu(Left), Tanh(right) of combined data

Combined Categories	ReLu	Tanh
Test Score	6.429523944854736	4.848769187927246
Test Accuracy	0.030927835032343864	0.10309278219938278

Table I-3.2-4 Test score and accuracy between ReLu and Tanh of combined data

Overall we found out that validation accuracy is higher than training accuracy after combined categories (Fig I-3.2-4). It seems like underfitting and it may be caused by the simplicity of features in validation data compared to the training data. To solve this problem we could try to increase the complexity of the model by using the more complex model, using more relevant features in the model, or experimenting with different hyperparameter settings.

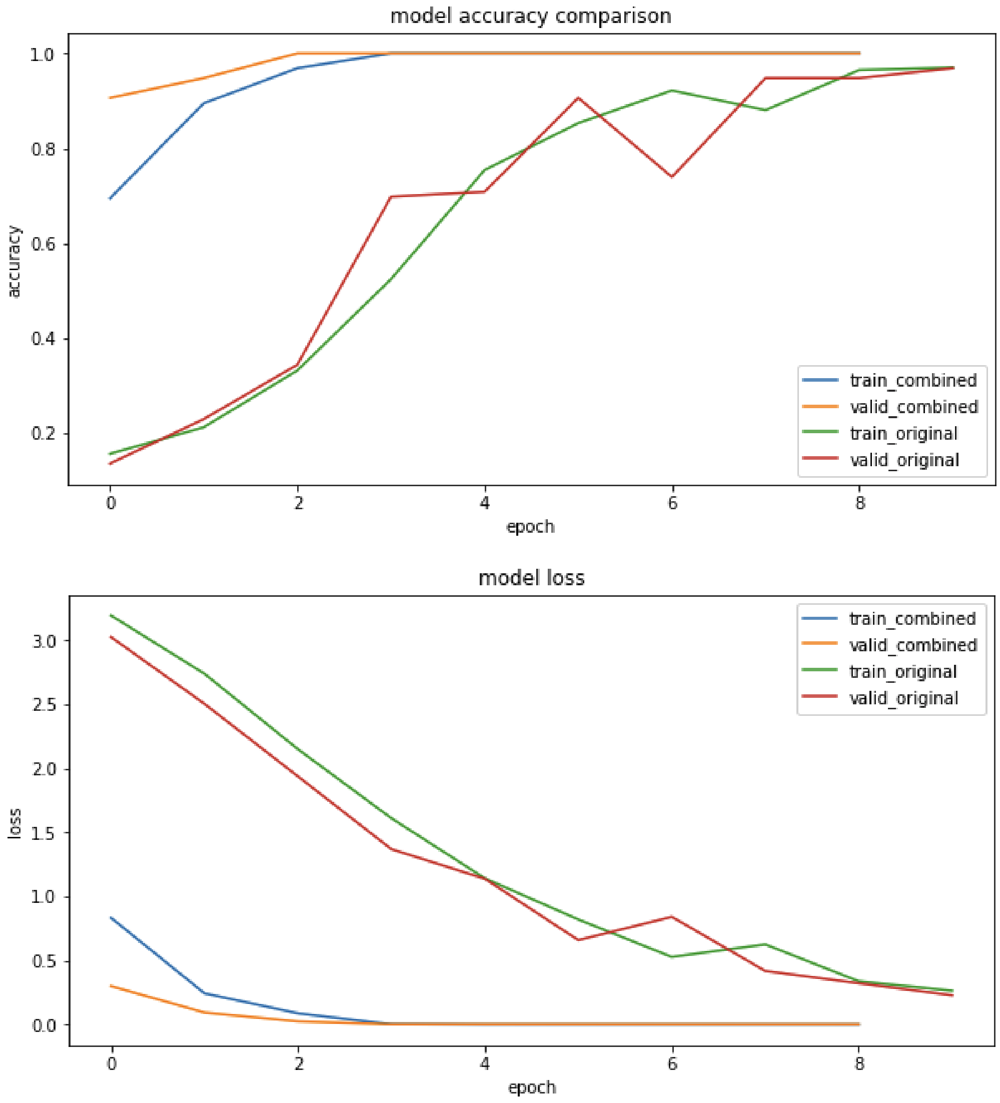


Fig I-3.2-4 combined and original resume datasets' RNN model accuracy and loss

Part II. Resume Dataset 2

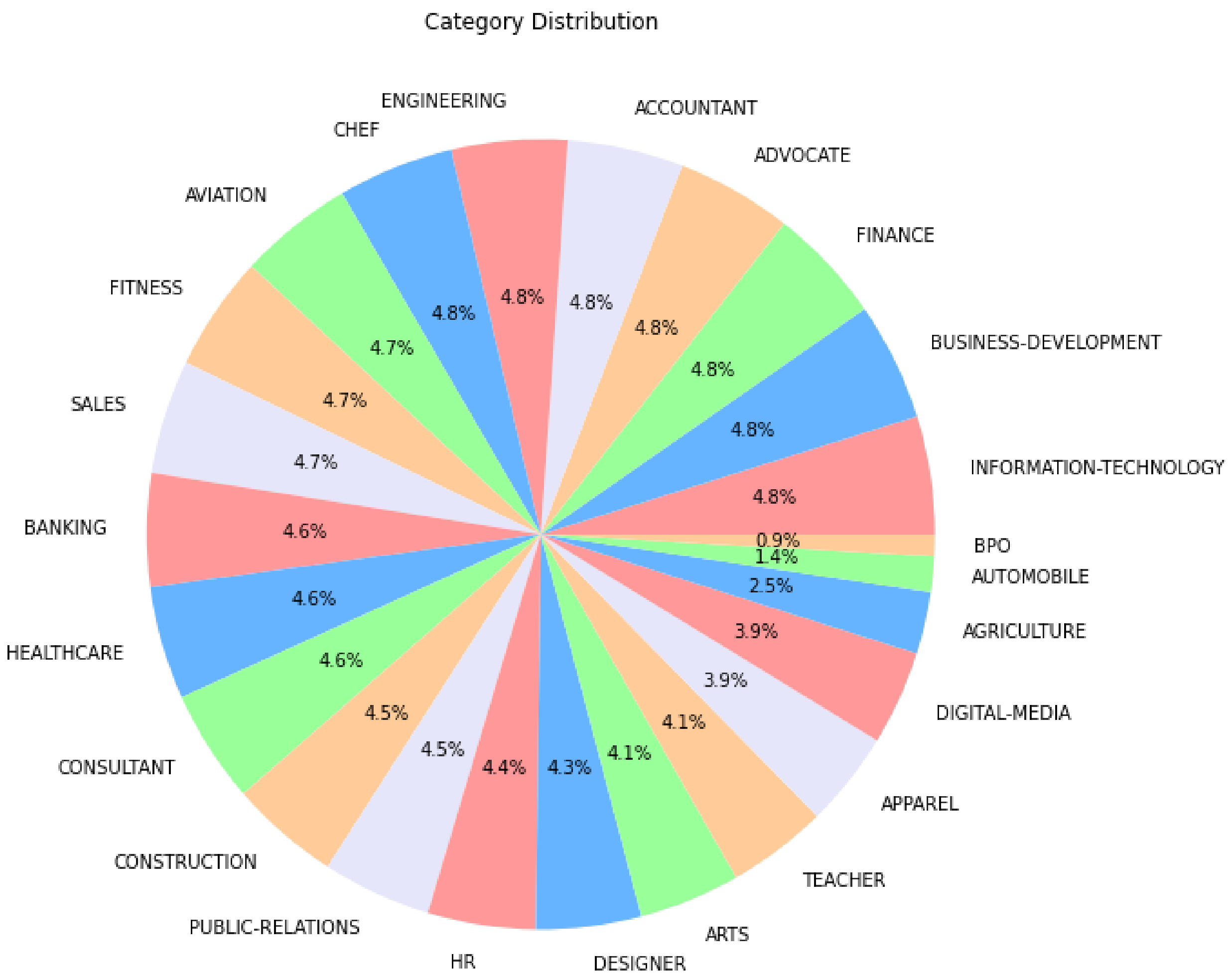
II-1. Method

(a) Dataset

The second dataset[2] we used is a collection of resume examples taken from livecareer.com. It contains 2400+ resumes in string format as well as PDF format. Inside the CSV file, columns consist of ID (Unique identifier and file name for the respective pdf), Resume_str (Contains the resume text only in string format), Resume_html (Contains the resume data in HTML format as a present while web scrapping), and Category (Category of the job the resume was used to apply). The categories column contains HR, Designer, Information-Technology, Teacher, Advocate, Business-Development, Healthcare, Fitness,

[2] https://www.kaggle.com/datasets/snehaanbhawal/resume-dataset

Agriculture, BPO, Sales, Consultant, Digital-Media, Automobile, Chef, Finance, Apparel, Engineering, Accountant, Construction, Public-Relations, Banking, Arts, Aviation.



(b) Data Cleaning and Preprocessing Steps

In our dataset, we aimed to have an average of 3% contribution from each job category, totaling at least 90 rows per category. However, the categories of ['AGRICULTURE', 'AUTOMOBILE', 'BPO'] had only 63, 36, and 22 rows, respectively. In order to ensure an equal amount of data for each category, we decided to drop these three job categories.

To clean the data, we initially looked at the "Resume_str" column which contained all the text of the resumes. However, we found that this column had a large amount of data and was difficult to clean. Therefore, we turned to the "Resume_html" column which contained HTML code. Using the Python Beautiful Soup library, we were able to extract the relevant information contained in the <li> elements. Finally, we performed a text cleansing process to remove any quotation marks, excess whitespace, URLs, etc.

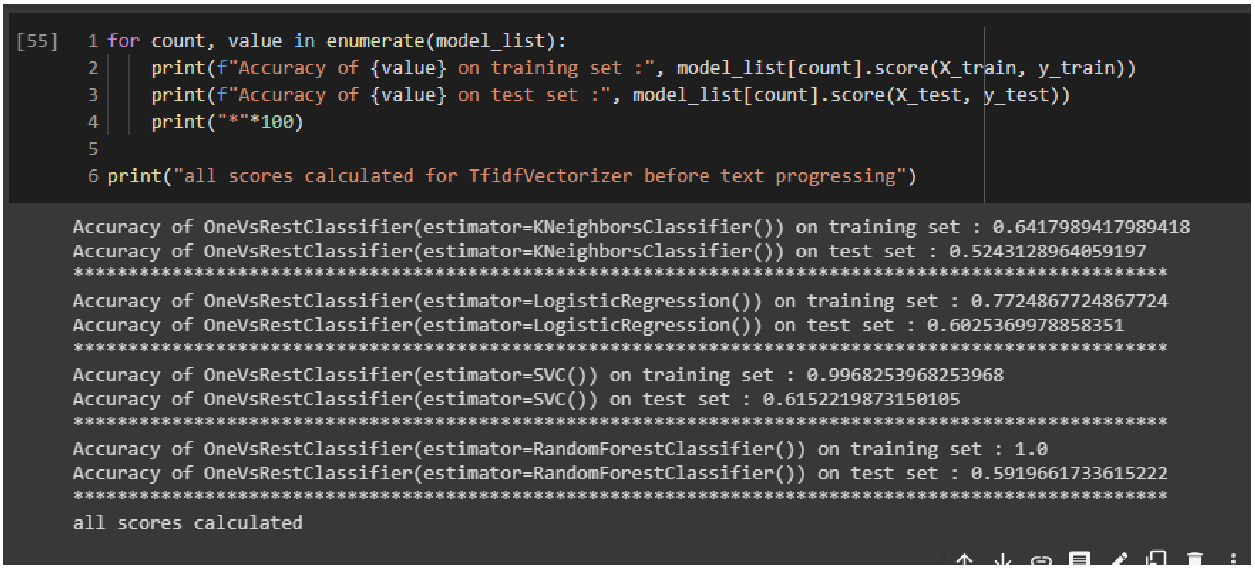
To investigate the impact of stopwords on our analysis, we used the spacy and NLTK libraries to identify and remove stopwords from our dataset. This was done as part of the preprocessing steps outlined above.

(c) Machine Learning Methods

We used KNN, Logistic Regression, SVC, and Random Forest Classifier to classify resumes. We employed the one-vs-the-rest (OvR) multiclass strategy for all of the models. To use text data in machine learning, we employed vectorizers to map words or phrases from the vocabulary to corresponding vectors of real numbers. CountVectorizer and TfidfVectorizer are two different vectorizers that can be used for this purpose. CountVectorizer converts text into a vector based on the frequency of each word that occurs in the entire text, while TfidfVectorizer converts a collection of raw documents into a matrix of TF-IDF features. One specific parameter on TfidfVectorizer is stop_words='english', which ignores unnecessary words.

II-2. Experiment:

In this dataset, we used two different vectorizers, CountVectorizer and TfidfVectorizer, to compare their performance. Additionally, we wanted to investigate the impact of text processing on the model's performance. Typically, natural language processing is used to filter words and remove stop words for text data like this, but the dataset has its own method of filtering words. We wanted to see if TfidfVectorizer performed better after our own text processing.

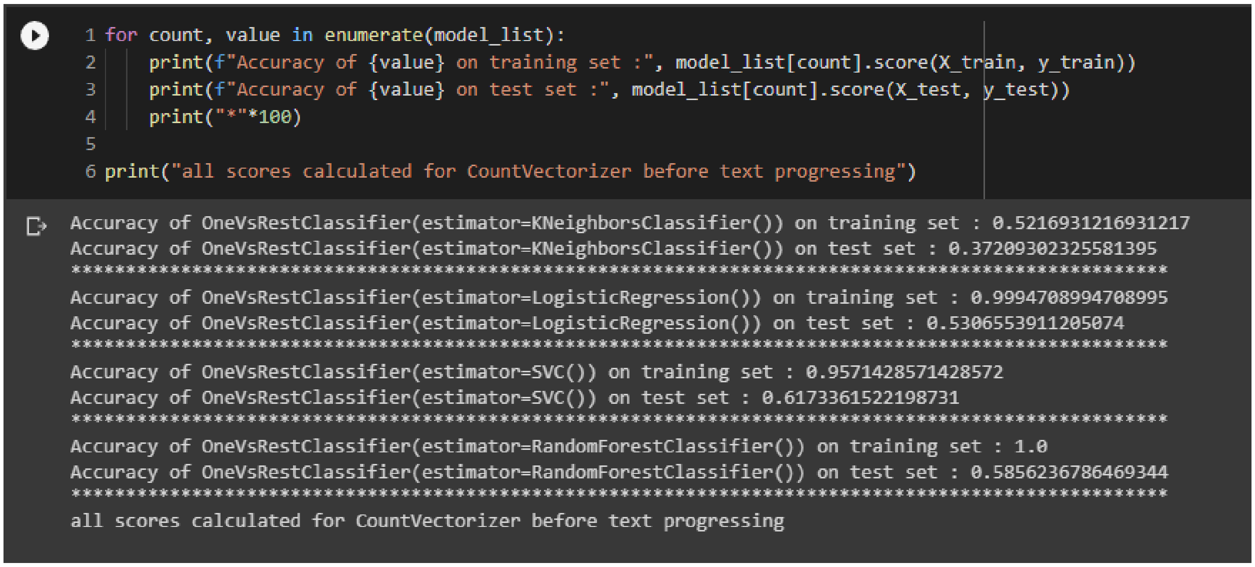


```
for count, value in enumerate(model_list):
    print(f"Accuracy of {value} on training set :", model_list[count].score(X_train, y_train))
    print(f"Accuracy of {value} on test set :", model_list[count].score(X_test, y_test))
    print("*"*100)

print("all scores calculated for TfidfVectorizer before text progressing")
```

```
Accuracy of OneVsRestClassifier(estimator=KNeighborsClassifier()) on training set : 0.6417989417989418
Accuracy of OneVsRestClassifier(estimator=KNeighborsClassifier()) on test set : 0.5243128964059197
****************************************************************************************************
Accuracy of OneVsRestClassifier(estimator=LogisticRegression()) on training set : 0.7724867724867724
Accuracy of OneVsRestClassifier(estimator=LogisticRegression()) on test set : 0.6025369978858351
****************************************************************************************************
Accuracy of OneVsRestClassifier(estimator=SVC()) on training set : 0.9968253968253968
Accuracy of OneVsRestClassifier(estimator=SVC()) on test set : 0.6152219873150105
****************************************************************************************************
Accuracy of OneVsRestClassifier(estimator=RandomForestClassifier()) on training set : 1.0
Accuracy of OneVsRestClassifier(estimator=RandomForestClassifier()) on test set : 0.5919661733615222
****************************************************************************************************
all scores calculated
```

Fig II-3.1 Using TfidfVectorizer without text processing

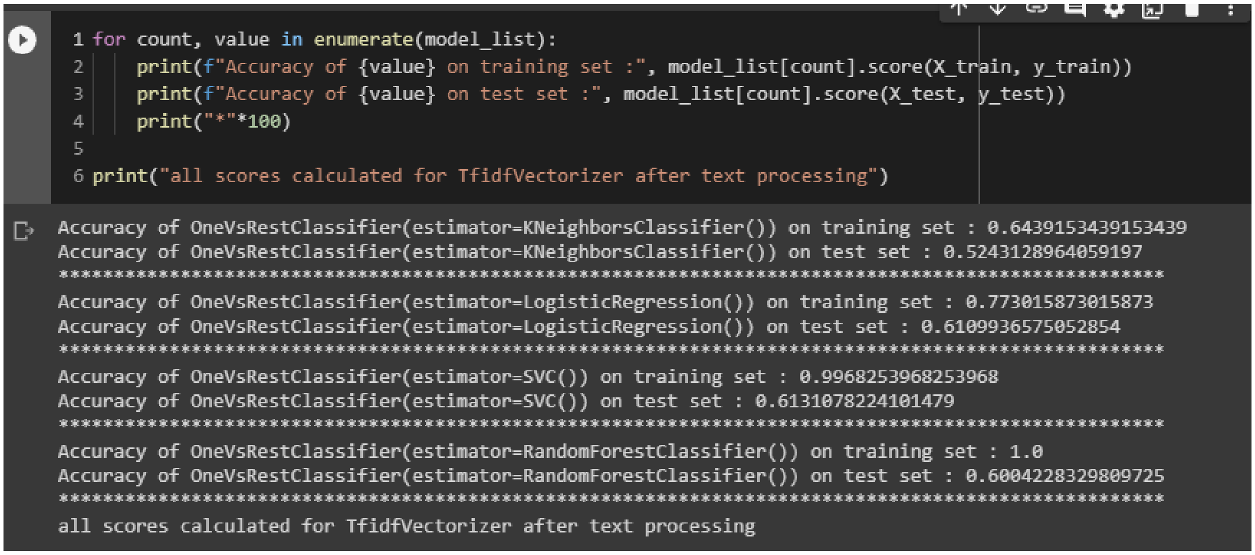


```
for count, value in enumerate(model_list):
    print(f"Accuracy of {value} on training set :", model_list[count].score(X_train, y_train))
    print(f"Accuracy of {value} on test set :", model_list[count].score(X_test, y_test))
    print("*"*100)

print("all scores calculated for CountVectorizer before text progressing")
```

```
Accuracy of OneVsRestClassifier(estimator=KNeighborsClassifier()) on training set : 0.5216931216931217
Accuracy of OneVsRestClassifier(estimator=KNeighborsClassifier()) on test set : 0.37209302325581395
****************************************************************************************************
Accuracy of OneVsRestClassifier(estimator=LogisticRegression()) on training set : 0.9994708994708995
Accuracy of OneVsRestClassifier(estimator=LogisticRegression()) on test set : 0.5306553911205074
****************************************************************************************************
Accuracy of OneVsRestClassifier(estimator=SVC()) on training set : 0.9571428571428572
Accuracy of OneVsRestClassifier(estimator=SVC()) on test set : 0.6173361522198731
****************************************************************************************************
Accuracy of OneVsRestClassifier(estimator=RandomForestClassifier()) on training set : 1.0
Accuracy of OneVsRestClassifier(estimator=RandomForestClassifier()) on test set : 0.5856236786469344
****************************************************************************************************
all scores calculated for CountVectorizer before text progressing
```

Fig II-3.2 Using CountVectorizer without text processing

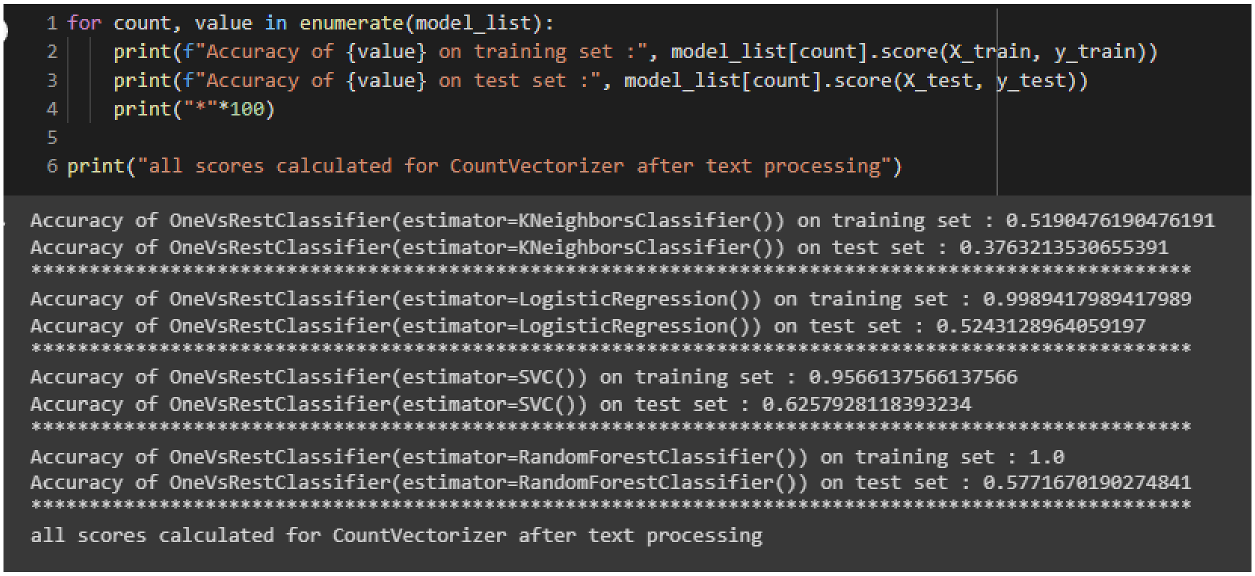


```
for count, value in enumerate(model_list):
    print(f"Accuracy of {value} on training set :", model_list[count].score(X_train, y_train))
    print(f"Accuracy of {value} on test set :", model_list[count].score(X_test, y_test))
    print("*"*100)

print("all scores calculated for TfidfVectorizer after text processing")
```

```
Accuracy of OneVsRestClassifier(estimator=KNeighborsClassifier()) on training set : 0.6439153439153439
Accuracy of OneVsRestClassifier(estimator=KNeighborsClassifier()) on test set : 0.5243128964059197
****************************************************************************************************
Accuracy of OneVsRestClassifier(estimator=LogisticRegression()) on training set : 0.773015873015873
Accuracy of OneVsRestClassifier(estimator=LogisticRegression()) on test set : 0.6109936575052854
****************************************************************************************************
Accuracy of OneVsRestClassifier(estimator=SVC()) on training set : 0.9968253968253968
Accuracy of OneVsRestClassifier(estimator=SVC()) on test set : 0.6131078224101479
****************************************************************************************************
Accuracy of OneVsRestClassifier(estimator=RandomForestClassifier()) on training set : 1.0
Accuracy of OneVsRestClassifier(estimator=RandomForestClassifier()) on test set : 0.6004228329809725
****************************************************************************************************
all scores calculated for TfidfVectorizer after text processing
```

Fig II-3.3 Using TfidfVectorizer after text processing



```
for count, value in enumerate(model_list):
    print(f"Accuracy of {value} on training set :", model_list[count].score(X_train, y_train))
    print(f"Accuracy of {value} on test set :", model_list[count].score(X_test, y_test))
    print("*"*100)

print("all scores calculated for CountVectorizer after text processing")
```

```
Accuracy of OneVsRestClassifier(estimator=KNeighborsClassifier()) on training set : 0.5190476190476191
Accuracy of OneVsRestClassifier(estimator=KNeighborsClassifier()) on test set : 0.3763213530655391
****************************************************************************************************
Accuracy of OneVsRestClassifier(estimator=LogisticRegression()) on training set : 0.9989417989417989
Accuracy of OneVsRestClassifier(estimator=LogisticRegression()) on test set : 0.5243128964059197
****************************************************************************************************
Accuracy of OneVsRestClassifier(estimator=SVC()) on training set : 0.9566137566137566
Accuracy of OneVsRestClassifier(estimator=SVC()) on test set : 0.6257928118393234
****************************************************************************************************
Accuracy of OneVsRestClassifier(estimator=RandomForestClassifier()) on training set : 1.0
Accuracy of OneVsRestClassifier(estimator=RandomForestClassifier()) on test set : 0.5771670190274841
****************************************************************************************************
all scores calculated for CountVectorizer after text processing
```

Fig II-3.4 Using CountVectorizer after text processing

II-3. Results and discussion:

From the results of vectorizers without text processing, we found that TfidfVectorizer performed well on KNN and Logistic Regression models but suffered from overfitting on SVC and Random Forest models. In comparison, the CountVector method showed underfitting on the KNN model and overfitting on the Logistic Regression, SVC, and Random Forest models. When we applied text processing to the vectorizers, we saw little change in the results. After text processing, TfidfVectorizer increased the test accuracy by about 1% for Logistic Regression and Random Forest models. However, text processing with CountVector actually decreased the test accuracy by about 1% for the Logistic Regression, SVC, and Random Forest models. In the future, we would like to get deeper into models overfitting.

Before this experiment, we were concerned about the cleanliness of the text input to the model, such as city names, stopwords, and other text-related factors that might affect the models. From this experiment, we found that TfidfVectorizer performs well with general data cleaning and doesn't require additional text regularization like lemmatization and stemming. Since we don't need to consider the frequency of words to categorize resumes, we can use TfidfVectorizer as a good vectorizer for training a resume screening model. KNN and Logistic Regression with the Onevsrest classifier will be our focus for further experimentation. In the future, we can create a frequency list of resume categories and combine it with job descriptions to use CountVector for a different type of resume screening model.

Experiment of Vectorizers without Text Processing

Vectorizer	Classifiers	KNN	Logistic Regression	SVC	RandomForest
TfidfVectorizer	Training Accuracy	0.64179	0.77248	0.99682	1.0

	Test Accuracy	0.52431	0.60253	0.61522	0.59196
CountVector	Training Accuracy	0.52169	0.99947	0.95714	1.0
	Test Accuracy	0.37209	0.53065	0.61733	0.58562

Experiment of Vectorizers with Text Processing

	Classifiers	KNN	Logistic Regression	SVC	RandomForest
TfidfVectorizer	Training Accuracy	0.64391	0.77301	0.99682	1.0
	Test Accuracy	0.52431	0.61099	0.61310	0.60042
CountVector	Training Accuracy	0.51904	0.99894	0.95661	1.0
	Test Accuracy	0.37632	0.52431	0.62579	0.57716

Conclusion

In this project with the two datasets containing strings. In the first dataset, we experimented with multiclass classification with KNN and RNN models. Additionally, by experimenting with processing an image of a resume, we explored the field of converting images into strings and combining them into the existing dataset. In this process, we learned text normalization for natural language processing like tokenization, removing and selecting necessary information.

In the second dataset, we learned the Python Beautiful Soup library to extract text from HTML data. We experimented with two different text vectorizers and text progressing. We learned text regularization methods like lemmatization and stemming to clean data. We learned the model is not affected by text progress on two vectorizers. Furthermore, the TfidfVectorizer is a good vectorizer for our resume screening.

While model building, we could experience how the parameters affect the results and how to manipulate datasets and fit them in the model. Also, we could experience designing and splitting datasets for models and checking the quality of those each dataset in depth.

Colab Notebooks:

Part 1

Part 2

Reference

Divya Mule, Samiksha Doke, Shakshi Navale, Prof S.K.Said. “RESUME SCREENING USING LSTM”, *International Research Journal of Engineering and Technology (IRJET)*, Volume: 09 Issue: 04 Apr 2022

“NLP: Text Generation through Bidirectional LSTM model” *medium,* 30 Jan. 2021, https://towardsdatascience.com/nlp-text-generation-through-bidirectional-lstm-model-9af29da4e520

“Quick Introduction to Bag-of-Words(Bow) and TF-IDF for Creating Features from Text”, *Analytics Vidhya*, 23 Dec. 2020, https://www.analyticsvidhya.com/blog/2020/02/quick-introduction-bag-of-words-bow-tf-idf/#:~:text=Bag%20of%20Words%20just%20creates,less%20important%20ones%20as%20well

“Resume Dataset.” Kaggle, 24 Feb. 2021, www.kaggle.com/datasets/gauravduttakiit/resume-dataset.